Lecture 8

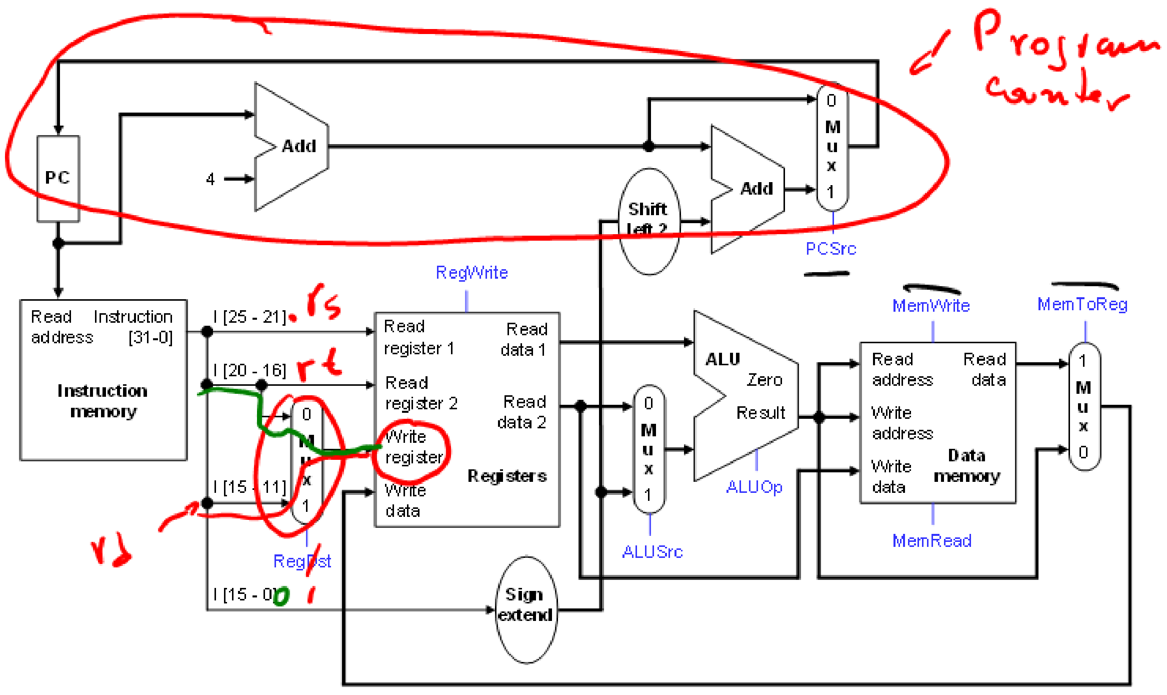
- Today
  - Finish single-cycle datapath/control path
  - Look at its performance and how to improve it. 2 2

2

now high tech! V2.9

2

The final datapath



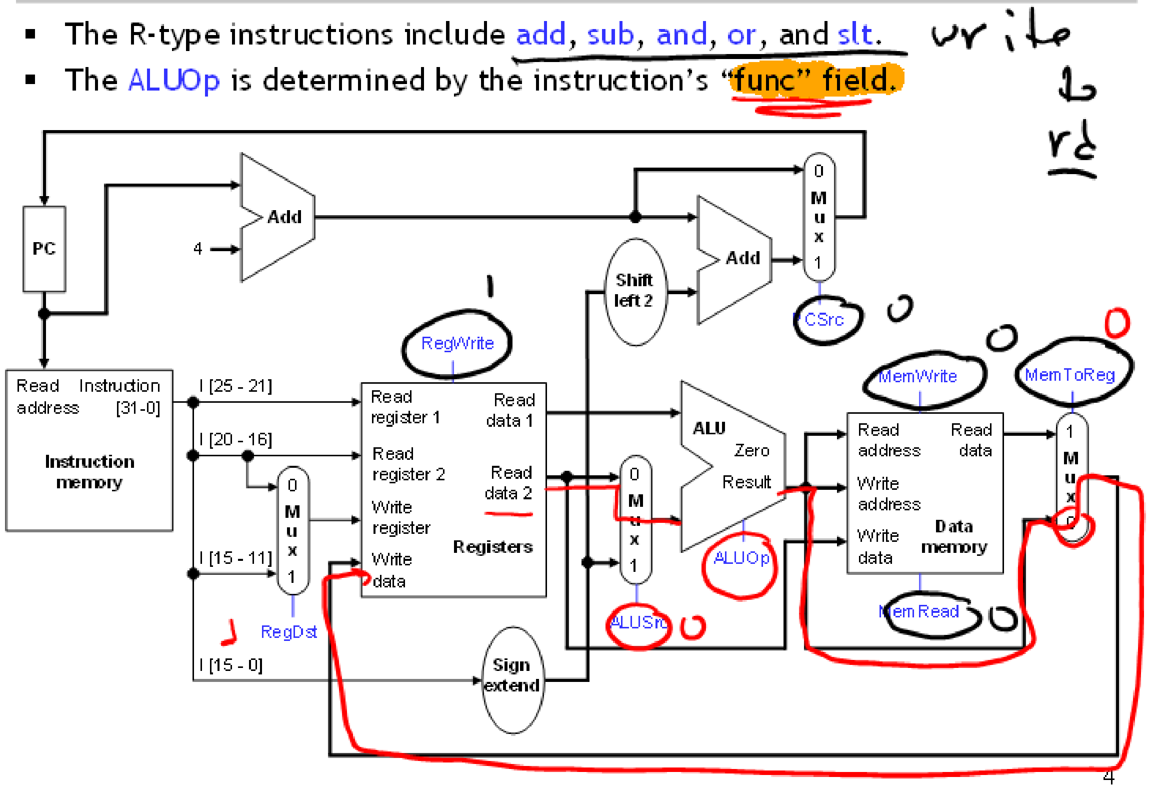
Control

- The control unit is responsible for setting all the control signals so that each instruction is executed properly.
  - The control unit's input is the 32-bit instruction word. +
  - The outputs are values for the blue control signals in the datapath.
- Most of the signals can be generated from the instruction opcode alone, and not the entire 32-bit word.
- To illustrate the relevant control signals, we will show the route that is taken through the datapath by R-type, lw, sw and beq instructions.

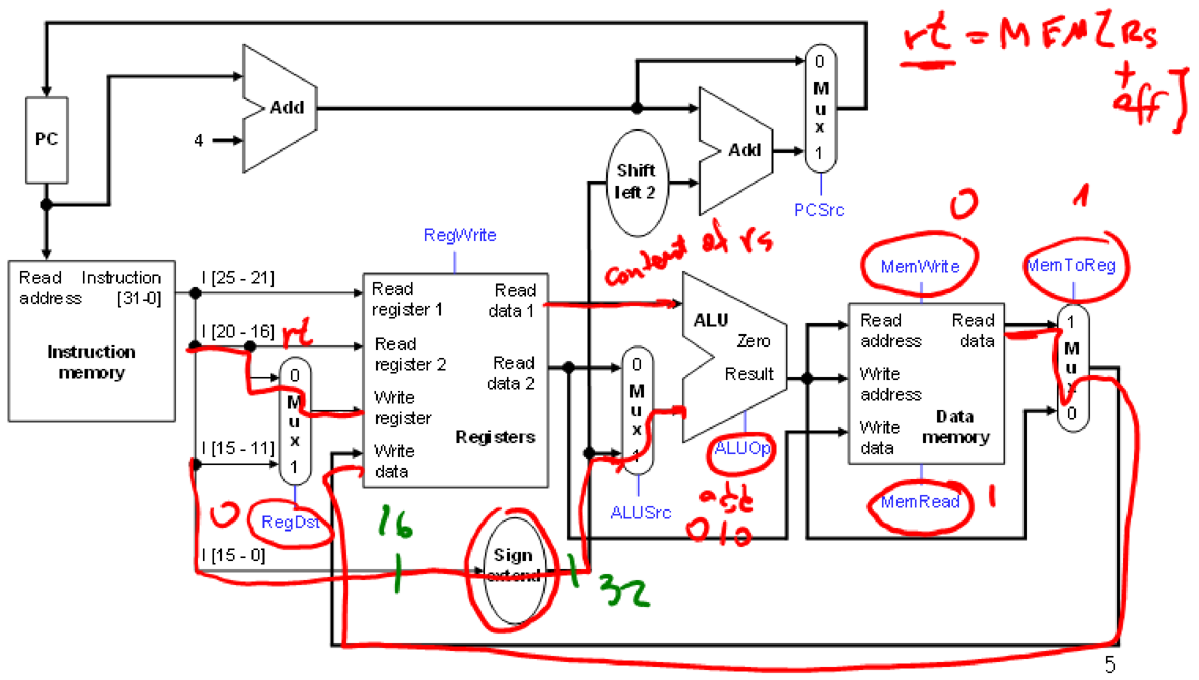
R-type instruction path

- The R-type instructions include add, sub, and, or, and slt.
- The ALUOp is determined by the instruction’s “func” field.

write
to
rd

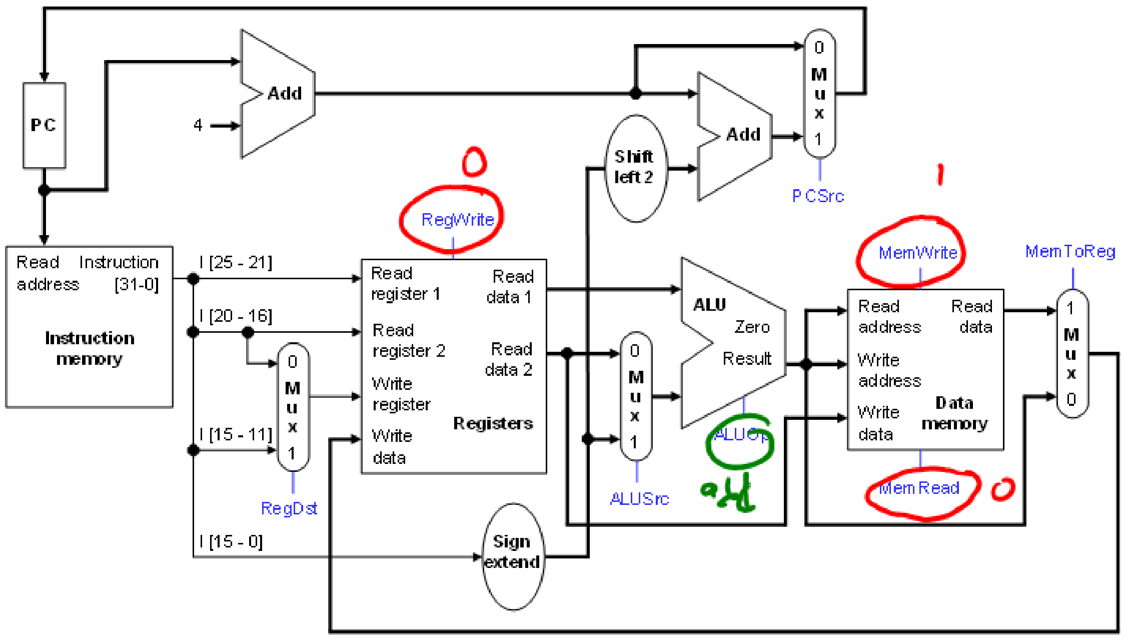


lw instruction path

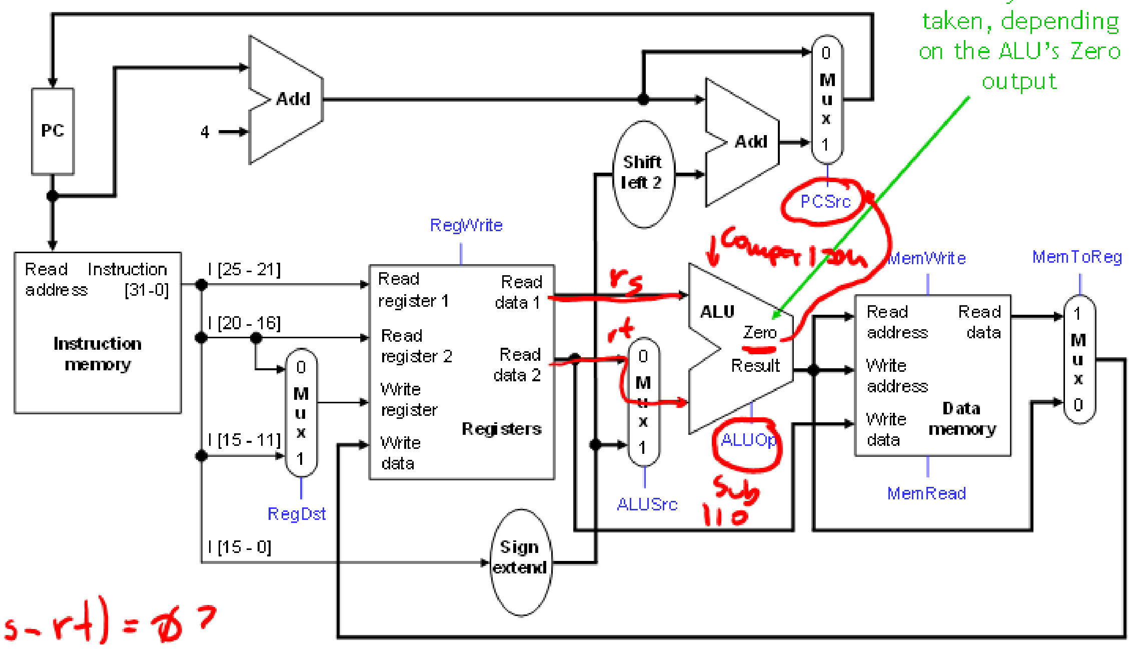


sw instruction path

- An example store instruction is sw $a0, 16($sp).
- The ALUOp must be 010 (add), again to compute the effective address.



beq instruction path



The branch may or may not be taken, depending on the ALU's Zero output

$(rs - rt) = \emptyset$?

Control signal table

PC Src

Operation	RegDst	RegWrite	ALUSrc	ALUOp	MemWrite	MemRead	MemToReg
add	1	1	0	010	0	0	0
sub	1	1	0	110	0	0	0
and	1	1	0	000	0	0	0
or	1	1	0	001	0	0	0
slt	1	1	0	111	0	0	0
lw	0	1	1	010	0	1	1
sw	X	0	1	010	1	0	X
beq	X	0	0	110	0	0	X

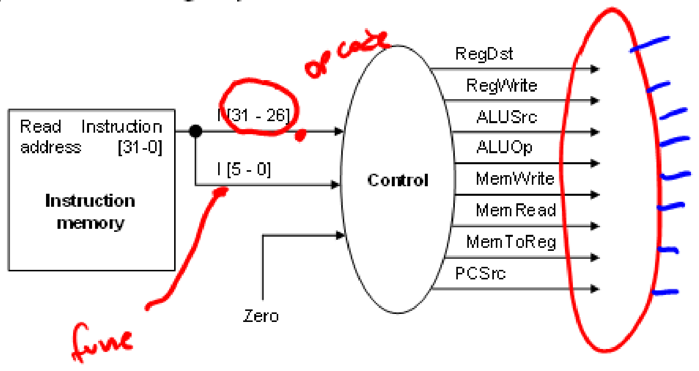
Rt

IL

- sw and beq are the only instructions that do not write any registers.
- lw and sw are the only instructions that use the constant field. They also depend on the ALU to compute the effective memory address.
- ALUOp for R-type instructions depends on the instructions' func field.
- The PCSrc control signal (not listed) should be set if the instruction is beq *and* the ALU's Zero output is true.

Generating control signals

- The control unit needs 13 bits of inputs.
  - Six bits make up the instruction's opcode.
  - Six bits come from the instruction's func field.
  - It also needs the Zero output of the ALU.
- The control unit generates 10 bits of output, corresponding to the signals mentioned on the previous page.
- You can build the actual circuit by using big K-maps, big Boolean algebra, or big circuit design programs.
- The textbook presents a slightly different control unit.



Summary of Single-Cycle Implementation

- A datapath contains all the functional units and connections necessary to implement an instruction set architecture.
  - For our single-cycle implementation, we use two separate memories, an ALU, some extra adders, and lots of multiplexers.
  - MIPS is a 32-bit machine, so most of the buses are 32-bits wide.
- The control unit tells the datapath what to do, based on the instruction that's currently being executed.
  - Our processor has ten control signals that regulate the datapath.
  - The control signals can be generated by a combinational circuit with the instruction's 32-bit binary encoding as input.
- Next, we'll see the performance limitations of this single-cycle machine and try to improve upon it.

Single-Cycle Performance

- We just saw a MIPS single-cycle datapath and control unit.
- Now we'll explore factors that contribute to a processor's execution time, and specifically at the performance of the single-cycle machine.
- Next time, we'll explore how to improve on the single cycle machine's performance using pipelining.

Three Components of CPU Performance

1 ✓ 1 1

$$\text{CPU time}_{X,P} = \underline{\text{Instructions executed}_P} * \boxed{\text{CPI}_{X,P}} * \boxed{\text{Clock cycle time}_X}$$

Program

Cycles Per Instruction

=1

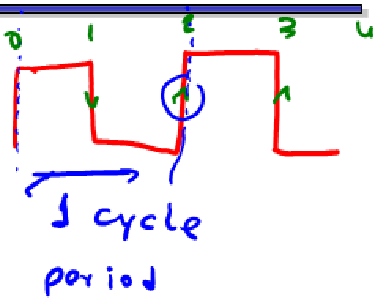
Instructions Executed

- Instructions executed:
  - We are not interested in the static instruction count, or how many lines of code are in a program.
  - Instead we care about the dynamic instruction count, or how many instructions are actually executed when the program runs. 2001
- There are three lines of code below, but how many instructions are actually executed? 3 static

```
          li    $a0, 1000           1 x 1
Ostrich:  sub   $a0, $a0, 1         1000 x
          bne   $a0, $0, Ostrich    1000 x
```

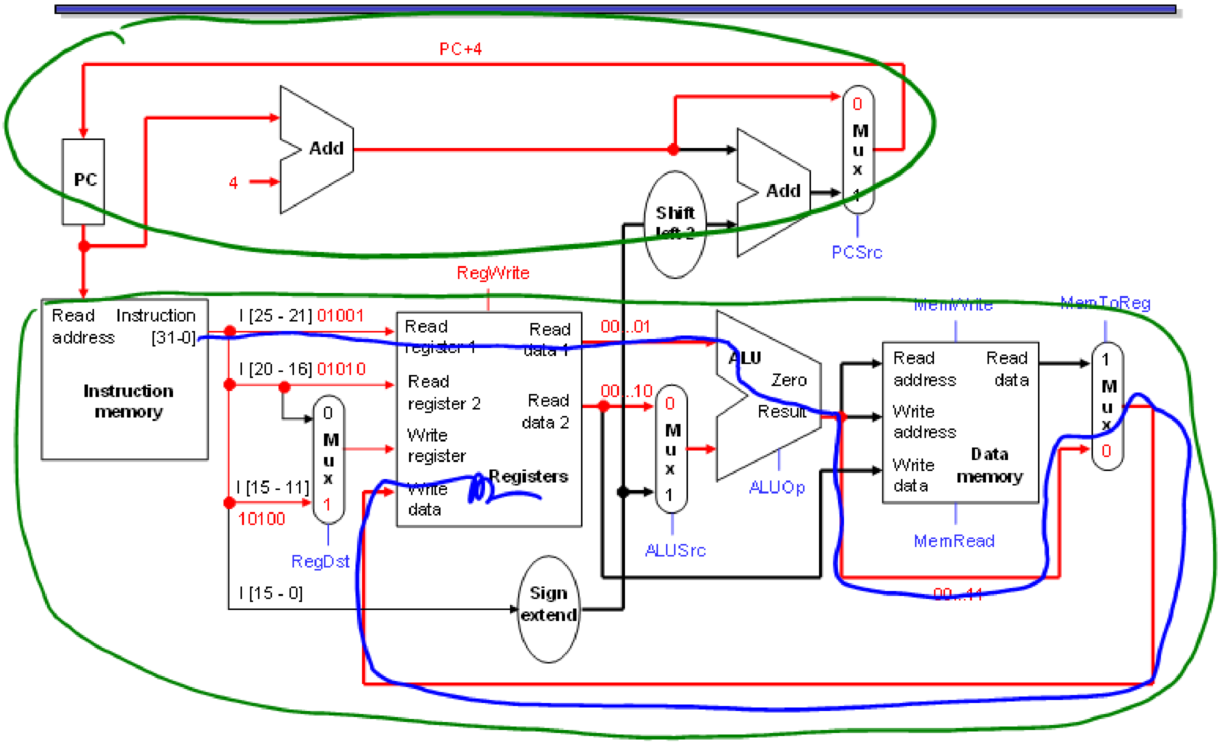
Clock cycle time

Clock rate = $\frac{1}{\text{period}}$



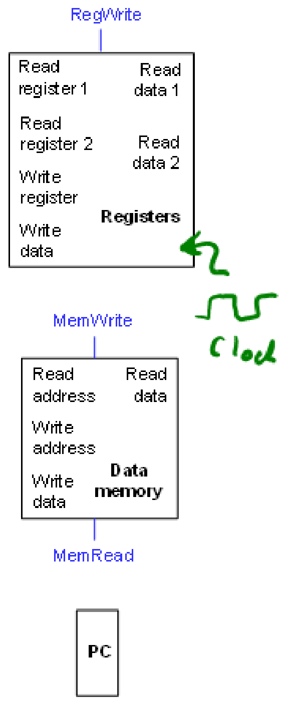
- One “cycle” is the minimum time it takes the CPU to do any work.
  - The clock cycle time or clock period is just the length of a cycle.
  - The clock rate, or frequency, is the reciprocal of the cycle time.
- Generally, a higher frequency is better.
- Some examples illustrate some typical frequencies.
  - A 500MHz processor has a cycle time of ? 2 ns
  - A 2GHz (2000MHz) CPU has a cycle time of just 0.5ns (500ps).

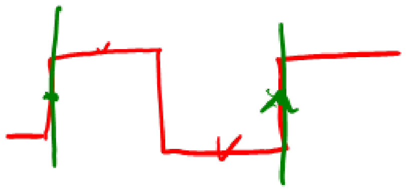
How the add goes through the datapath



Edge-triggered state elements

- In an instruction like add $t1, $t1, $t2, how do we know $t1 is not updated until *after* its original value is read?
- We'll assume that our state elements are positive edge triggered, and are updated only on the positive edge of a clock signal.
  - The register file and data memory have explicit write control signals, RegWrite and MemWrite. These units can be written to only if the control signal is asserted *and* there is a positive clock edge.
  - In a single-cycle machine the PC is updated on each clock cycle, so we don't bother to give it an explicit write control signal.





The datapath and the clock

1. On a positive clock edge, the PC is updated with a new address.
2. A new instruction can then be loaded from memory. The control unit sets the datapath signals appropriately so that
   - registers are read, ✓
   - ALU output is generated, ✓
   - data memory is read or written, and ✓
   - branch target addresses are computed. ✓

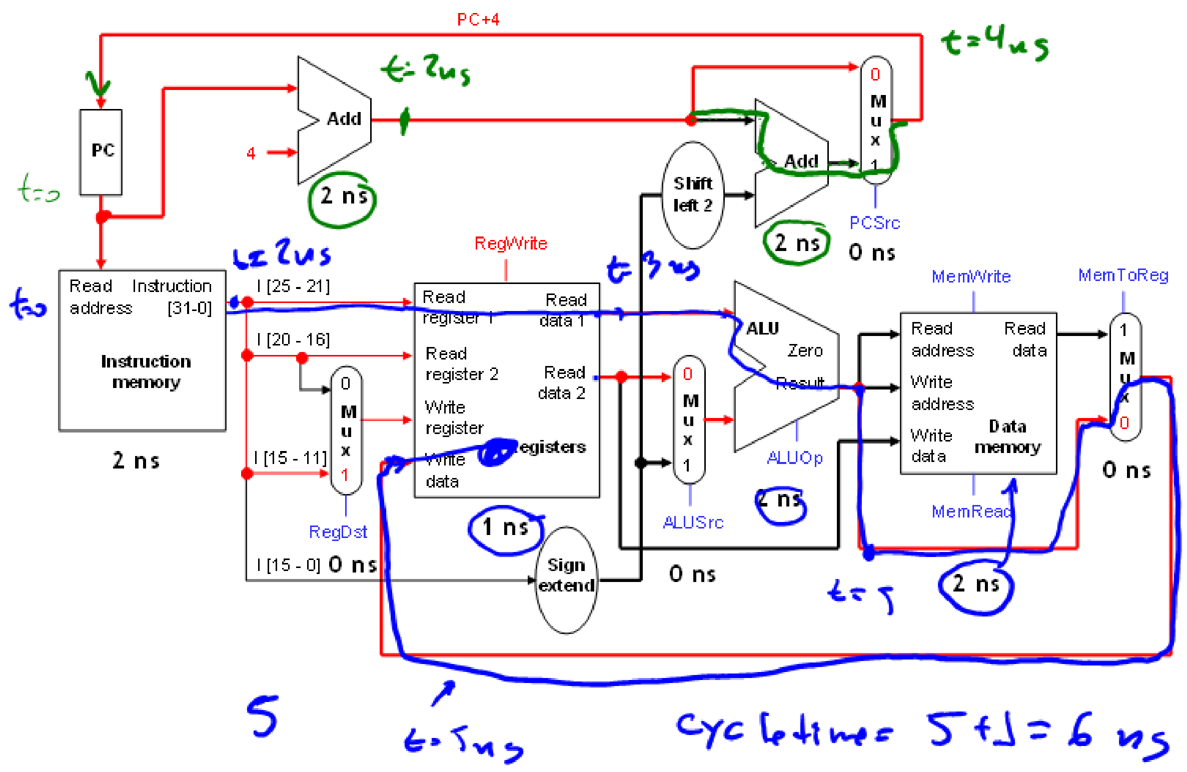
1 cycle

3. Several things happen on the *next* positive clock edge.
   - The register file is updated for arithmetic or lw instructions.
   - Data memory is written for a sw instruction. ✓
   - The PC is updated to point to the next instruction. ✓

- In a single-cycle datapath everything in Step 2 must complete within one clock cycle, before the next positive clock edge.

How long is that clock cycle?

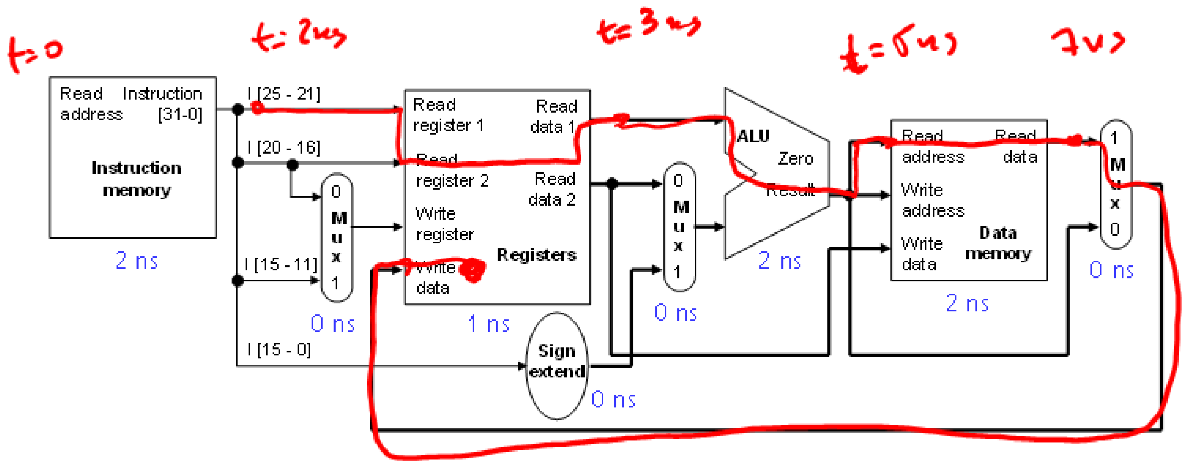
Compute the longest path in the add instruction



The slowest instruction…

- If all instructions must complete within one clock cycle, then the cycle time has to be large enough to accommodate the *slowest* instruction.
- For example, lw $t0, -4($sp) is the slowest instruction needing 8ns.
  - Assuming the circuit latencies below.

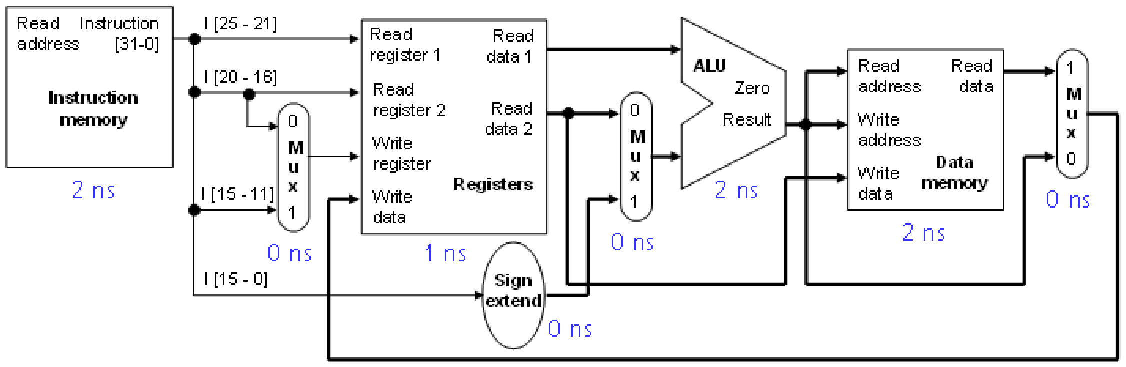
Cycle time: 7+1= 8ns



The slowest instruction…

- If all instructions must complete within one clock cycle, then the cycle time has to be large enough to accommodate the *slowest* instruction.
- For example, lw $t0, -4($sp) needs 8ns, assuming the delays shown here.

reading the instruction memory	2ns	
reading the base register $sp	1ns	
computing memory address $sp-4	2ns	8ns
reading the data memory	2ns	
storing data back to $t0	1ns	

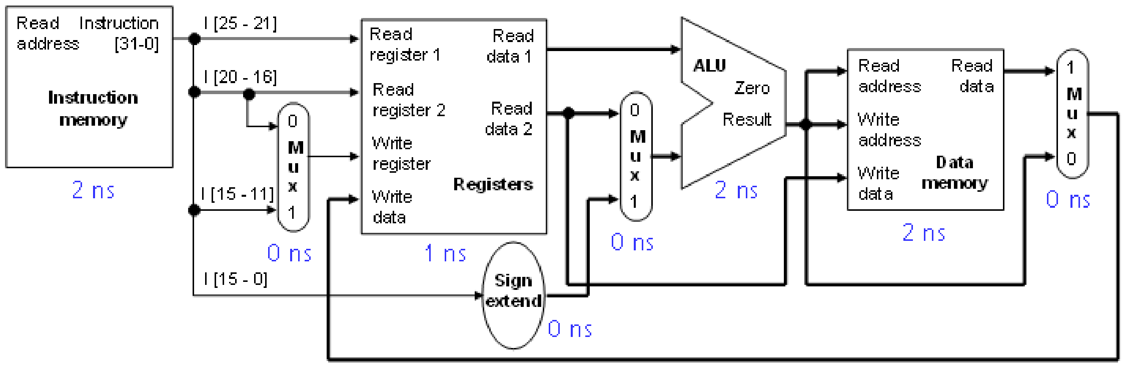


...determines the clock cycle time

- If we make the cycle time 8ns then *every* instruction will take 8ns, even if they don't need that much time.
- For example, the instruction add $s4, $t1, $t2 really needs just 6ns.

reading the instruction memory	2 ns
reading registers $t1 and $t2	1 ns
computing $t1 + $t2	2 ns
storing the result into $s0	1 ns

6 ns



Performance of Single-cycle Design

CPU time$_{X,P}$ = ~~Instructions executed$_P$~~ N * CPI$_{X,P}$ * $\underline{\text{Clock cycle time}_X}$

$\text{CPU}_{\text{time}} = N \times 1 \times 8 \times 10^{-9}$

$N = 1$ billion instrs $\Rightarrow$ 8 seconds

How bad is this?

- With these same component delays, a sw instruction would need 7ns, and beq would need just 5ns.
- Let's consider the gcc instruction mix from p. 189 of the textbook.

Instruction	Frequency
Arithmetic	48%
Loads	22%
Stores	11%
Branches	19%

- With a single-cycle datapath, each instruction would require 8ns.
- But if we could execute instructions as fast as possible, the average time per instruction for gcc would be:

$$(48\% \times 6\text{ns}) + (22\% \times 8\text{ns}) + (11\% \times 7\text{ns}) + (19\% \times 5\text{ns}) = 6.36\text{ns}$$

- The single-cycle datapath is about 1.26 times slower!

It gets worse…

- We've made very optimistic assumptions about memory latency:
  - Main memory accesses on modern machines is >50ns.
    - For comparison, an ALU on an AMD Opteron takes ~0.3ns. 3.33GHz
- Our worst case cycle (loads/stores) includes 2 memory accesses
  - A modern single cycle implementation would be stuck at <10Mhz.
  - Caches will improve common case access time, not worst case.
- Tying frequency to worst case path violates first law of performance!!
  - "Make the common case fast" (we'll revisit this often)

Summary

- Performance is one of the most important criteria in judging systems.
  - Here we'll focus on Execution time.
- Our main performance equation explains how performance depends on several factors related to both hardware and software.

$$\text{CPU time}_{X,P} = \text{Instructions executed}_{P} * \text{CPI}_{X,P} * \text{Clock cycle time}_{X}$$

- It can be hard to measure these factors in real life, but this is a useful guide for comparing systems and designs.
- A single-cycle CPU has two main disadvantages.
  - The cycle time is limited by the worst case latency.
  - It isn't efficiently using its hardware.
- Next time, we'll see how this can be rectified with pipelining.